Using Triggered Operations to Offload Collective Communication Operations

K. Scott Hemmert[1], Brian Barrett[1], and Keith D. Underwood[2]

[1] Sandia National Laboratries*
P.O. Box 5800, MS-1110
Albuquerque, NM, 87185-1110
kshemme@sandia.gov, bwbarre@sandia.gov
[2] Intel Corporation
Hillsboro, OR, USA
keith.d.underwood@intel.com

Abstract. Efficient collective operations are a major component of application scalability. Offload of collective operations onto the network interface reduces many of the latencies that are inherent in network communications and, consequently, reduces the time to perform the collective operation. To support offload, it is desirable to expose semantic building blocks that are simple to offload and yet powerful enough to implement a variety of collective algorithms. This paper presents the implementation of barrier and broadcast leveraging triggered operations — a semantic building block for collective offload. Triggered operations are shown to be both semantically powerful and capable of improving performance.

1 Introduction

Although the vast majority of data volume that is transferred within science and engineering applications is in relatively localized, point to point communications, these applications also include some number of global communications known as collectives. Many collective communications are inherently less scalable, as they involve communications all the way across the machine and contributions from every node. As system sizes increase, it becomes increasingly difficult to implement fast collectives across the entire system. One approach to improving collective performance is to offload collective operations to the network.

Many prior approaches to offloading collective operations have offloaded the entire collective operation, including the communication setup and computation [1]. While this eliminates the host overhead, it creates a more complicated offload function that is harder to adapt over time. As an alternative, a similar level of offload can be achieved with more elementary building blocks that are both easier to implement in hardware and less subject to change. When building

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

blocks are provided, the host library (e.g., MPI) is able to more readily adopt new collective algorithms as they are being developed. Alternatively, the host can more easily tune the algorithm based on the size of the system, the layout of the job on the system, and the size of the collective.

Portals 4 [2] introduced a set of semantic building blocks that included triggered operations and counting events that were explored for `MPI_ALLREDUCE` in [3]. Triggered operations allow an application to schedule a new network operation to occur in the future when a counting event reaches a specified threshold. This paper illustrates the breadth of triggered operations for implementing collective algorithms ranging from tree based barriers to dissemination barriers to bulk data broadcasts. Simulation results show the performance improvements that can be achieved using offload through triggered operations.

2 Related Work

Offload of collective operations has been an active area of research for many years. Custom engineered systems like the Cray T3D provided hardware barrier synchronization [4] and IBM's BG/L provided a dedicated collective network. Similarly, research into hardware support for collective operations on commodity hardware began in the mid-1990's [5]. This work became more prevalent with the arrival of programmable network interfaces like Myrinet and Quadrics. Barrier [1, 6] and broadcast [7] are particularly popular targets.

Offloading collective operations onto a Myrinet NIC requires significant enhancements to the control program running on the NIC processor. Because this requires significant effort for each collective operation offloaded, mechanisms to provide more dynamic offloaded capability were proposed [8]. Unlike Myrinet, the Quadrics Elan network supported a user-level thread on the NIC. Because this user-level thread has direct access to the address space of the process that created it, it is easier to create extended functionality to offload collectives. The programming environment for the Elan adapters provides some key functionality. For example, Elan event functions can increment a counter by a user specified amount when an operation, such as a DMA transfer, completes. Events could be chained to allow the triggering of one event to trigger others. Elan events are very similar to the counting events that were added to the Portals [2] API.

3 Triggered Operations in Portals 4

Triggered operations and counting events were introduced into Portals 4 [2] as semantic building blocks for collective communication offload. Triggered operations provide a mechanism through which an application can schedule message operations that initiate when a condition is met. Triggered versions of each of the Portals data movement operations were added (e.g., `PtlTriggeredPut()`, `PtlTriggeredGet()`, and `PtlTriggeredAtomic()`) by extending the argument list to include a counting event on which the operation will trigger and a threshold at which it triggers. In turn, counting events are the lightweight semantic

provided to track the completion of network operations. Counting events are opaque objects containing an integer that can be allocated, set to a value, or incremented by a value through the Portals API. In addition, they can be attached to various Portals structures and configured to count a variety of network operations, such as the local or remote completion of a message as well as the completion of incoming operations on a buffer (e.g., the completion of a `PtlPut()` or `PtlAtomic()` to a local buffer).

A triggered operation is issued by the application and then initiated by the network layer when a counting event reaches a threshold. Through careful use of counting events and triggered operations, an almost arbitrary sequence of network operations can be setup by the application and then allowed to progress asynchronously. A discussion of how reduction operations can be implemented using triggered operations is presented in [3].

4 Evaluation Methodology

The Structural Simulation Toolkit (SST) v2.0 [9] was used to simulate both host-based and offloaded versions of several collective algorithms. SST provides a component-based simulation environment, designed for simulating large-scale HPC environments. It simultaneously provides both cycle-accurate and event-based simulation capabilities. Here, we present both the algorithms simulated and a description of the parameters used for simulation.

4.1 Collective Algorithms

Three barrier algorithms were simulated for both a host based and a triggered operation based implementation. The first algorithm was a binomial tree (not shown) with the experimentally determined optimal radix chosen for both the host and the triggered cases. The tree algorithm is similar to what was explored for Allreduce in prior work [3]. The second algorithm used was the recursive doubling algorithm (also not shown), which is a simplified variant of the Allreduce in [3], since no data movement is required.

The final algorithm explored is the dissemination barrier [10]. In the radix-2 version, the dissemination barrier has a series of rounds, R, where each node, N, sends a message to node $(N + 2^R)$ mod P. A message in a given round can only be sent after messages for all prior rounds have been received. Because some nodes can proceed through the rounds faster than others, a node must receive a specific set of messages before proceeding. This is synonymous with receiving the message for this round and having completed the previous round, which is how the algorithm in Figure 1 is structured. Figure 1 is also extended to show a higher radix dissemination barrier algorithm.

A binomial tree algorithm is used for broadcast. Figure 2 shows how triggered operations can be leveraged for a rendezvous style protocol implementing a tree. At communicator creation, each node creates a descriptor to receive messages from their "parent" in the tree. When the collective is initiated, children

```
//Round 0 message from self when we enter
for (j = 1; j < radix; j++) PtlPut(user_md_h, (id+j) % num_nodes, 0);
//Signal round 1. Only receive radix−1 messages and not signal from previous round
PtlTriggeredCTInc(level_ct_hs[1], 1, level_ct_hs[0], radix−1);
PtlTriggeredCTInc(level_ct_hs[0], −(radix−1), level_ct_hs[0], radix−1);
for (i = 1, level = 0x2 ; level < num_nodes ; level <<= log2(radix), ++i) {
    for (j = 0; j < (radix−1); ++j) {
        remote = (id + level + i) % num_nodes;
        // Start round i when input from round i − 1 peer arrives and
        // communication to round i − 1 completes
        PtlTriggeredPut(md_h, remote, i, level_ct_hs[i], radix);
    }
    //Signal round i+1 that round i (and all previous rounds) is done
    PtlTriggeredCTInc(level_ct_hs[i+1], 1, level_ct_hs[i], radix);
    //Clean−up this iteration
    PtlTriggeredCTInc(level_ct_hs[i], −radix, level_ct_hs[i], radix);
}
// wait for completion and clean up last level
PtlCTWait(level_ct_h[levels], 1);
PtlTriggeredCTInc(level_ct_hs[levels], −1, level_ct_hs[levels], 1);
```

Fig. 1. Pseudo-code for the triggered dissemination barrier algorithm

determine who their parent will be based on the root and issue a triggered get. When the data is available in the local buffer, the parent notifies the child, which increments the counting event that releases the triggered get. The algorithm is pipelined by issuing multiple triggered gets with offsets that trigger at different count thresholds. Short messages are sent using a puts into the bounce buffer, with a user-level copy on completion.

4.2 Simulation Model

The collective operation simulations utilize a cycle-based router and network model combined with an event driven model of the network interface and the host. A torus network of up to 32K nodes ($32 \times 32 \times 32$) was simulated. Simulations were run with and without simulated OS interference to determine the success of offloaded implementations in eliminating noise. The router simulation matched those used in earlier simulations [9]. In contrast, the node was modeled as a simple state machine. Message insertion rate, delays for copying data to the NIC, and delays associated with memory copies were all modeled as interrelated occupancies in a queuing model. The NIC used a similar set of occupancies to model the NIC level operations and fed data (packets) to the router model.

Key parameters were modeled for both the NIC and host processing times: bus delays, delays through the NIC, occupancy in the receive processing logic, and memory latencies. Parameters that were used corresponded to network la-

```
if (my_root == my_id) {
  /* Notify children that all chunks are ready */
  for (j = 0 ; j < msg_size ; j += chunk_size)
    for (i = 0 ; i < num_children ; ++i) PtlPut(bounce_md, 0, 0, 0, child[i], 0);
} else {
  /* iterate over chunks */
  for (offset = 0 ; offset < msg_size ; offset += chunk_size) {
    /* when a chunk is ready, issue get. Local and remote offset are the same */
    PtlTriggeredGet(out_md, offset, chunk_size, my_root, offset,
                    bounce_ct, j / chunk_size);
    /* then when the get is completed, send ready notice to children */
    for (i = 0 ; i < num_children ; ++i)
      PtlTriggeredPut(bounce_md, 0, 0, 0, child[i], 0, out_md_ct, offset / chunk_size);
  }
  /* reset 0−byte put received counter */
  PtlTriggeredCTInc(bounce_ct, −count, bounce_ct, count);
}
/* wait for children gets */
if (num_children > 0) PtlCTWait(out_me_ct, count);
/* wait for local gets to complete */
else PtlCTWait(out_md_ct, count);
```

Fig. 2. Pseudo-code for the long message triggered binomial tree broadcast

tencies of 1 μs and 1.5 μs and are shown in Table 1. The one way message rates are limited by the highest latency, unpipelined processing stage. Since the hardware stages are pipelined, the message rate limiter is the software, which yields 5.7 million messages per second (Mmsgs/s) and 3.3 Mmsgs/s for 1 μs and 1.5 μs latency, respectively. The rate at which the NIC can issue triggered operations is limited by the 8 flit header and the 500 MHz clock to yield 62.5 Mmsgs/s (once the operations are queued on the NIC). To enqueue triggered operations, the software faces the same limitations as message transmits; therefore, triggered operations can be enqueued at 10 Mmsgs/s and 5 Mmsgs/s, based on the 100 ns and 200 ns TX software delays, respectively. As a final parameter, setup time for the collective operation — time needed by MPI before communication starts to setup the algorithm — is set to 200 ns. In addition to the baseline simulations, we run simulations representing the impact of OS noise. One "long noise at infrequent interval" signature (25 μs at 1 KHz) from a previous study [11] is used to represent OS noise.

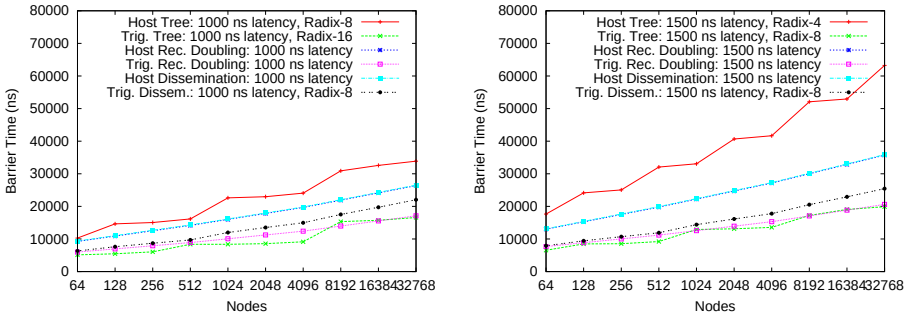
5 Results

Figure 3 compares the performance of the three barrier algorithms with 1000 ns and 1500 ns latency for both host and triggered implementations. The triggered implementation has over a 2× advantage that is larger at higher latency, since

Table 1. Summary of simulation parameters

Msg Latency	1000 ns	1500 ns	Msg Latency	1000 ns	1500 ns
TX Software Delay	100 ns	200 ns	RX Software Delay	175 ns	300 ns
TX Bus Delay	200 ns	300 ns	RX Bus Delay	200 ns	300 ns
TX NIC Delay	75 ns	100 ns	RX NIC Delay	150 ns	200 ns
Memory Latency	100 ns	100 ns	Read over Bus	400 ns	500 ns

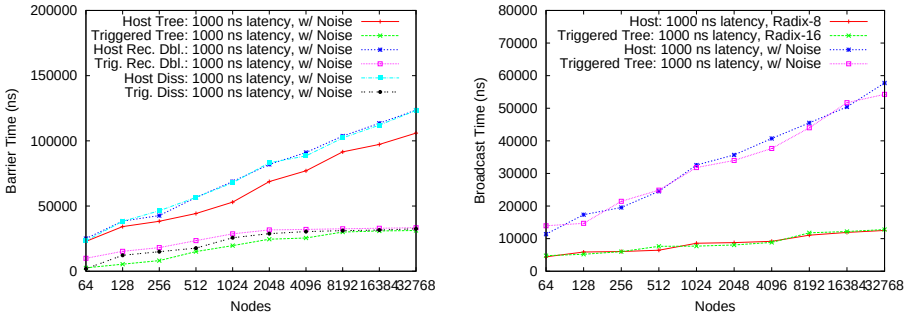
the triggered operations experience lower *effective* latency and higher *effective* message rate when they actually issue from the NIC. Much of the real latency and message posting overheads are overlapped with other communications for triggered operations (i.e., they happen before the host implementation is free to initiate the messages). Note that we used a Radix-8 implementation for the triggered dissemination barrier, which gave it a significant advantage over the very similar recursive doubling algorithm using Radix-2. Radix-8 results for the host variant (not shown) were far worse than Radix-2.



1000 ns latency 1500 ns latency

Fig. 3. Simulated barrier time

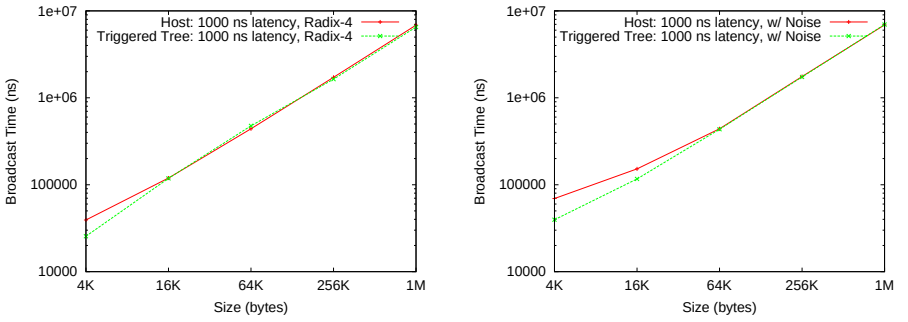
Figure 4 presents results of simulations with noise. While the barrier time increases substantially for host based implementations and shows growing impact as the number of nodes increases, the barrier time for implementations using triggered operations shows more modest impact from noise and the noise impact levels off at large node counts. In addition, note that the introduction of noise changes the "right" algorithm to use. Both dissemination and recursive doubling algorithms require processing by every node at every stage, but a tree based algorithm uses logarithmically fewer nodes at each stage. This carries over to triggered operations, since all nodes spend a significant amount of time injecting messages in the dissemination and recursive doubling barrier algorithms, but most nodes inject few messages in the binomial tree.



Barrier with Noise | Small Broadcast Performance

Fig. 4. Simulated time with Noise

Broadcast performance is presented in Figure 4 for small messages (8 bytes) and in Figure 5 for a sweep over larger messages at 4096 nodes. At small messages, the triggered operations provide a 15–20% performance improvement over host based algorithms, in addition to substantially less noise sensitivity. At larger message sizes, however, broadcast using triggered operations has a smaller performance and noise sensitivity advantage. Serialization delay to transfer data dominates both noise and processor overheads, which can be seen by the convergence of the host and triggered results in Figure 5. The triggered technique shows promise for non-blocking collectives, however, as it offers similar performance to host-based collectives with minimal processor overhead after an initial setup period. The use of a multi-post triggered interface to save round-trip communication with the NIC when setting up the messaging pipeline would further reduce the (small) processor overhead experienced for triggered bcasts of large messages and will be examined in future work.



Broadcast size sweep | Broadcast size sweep Noise

Fig. 5. Sweep of broadcast size with and without noise

6 Conclusions

This paper has illustrated that triggered operations leveraging counting events are semantically sufficient to implement a variety of collective algorithms. Pseudo-code was shown for a rendezvous-like functionality for long broadcasts and pseudo-code was shown for higher radix dissemination barriers. Collectives based on triggered operations are shown to be both higher performing (by over 2×) and more resistant to interference from system noise.

References

1. Buntinas, D., Panda, D.K., Sadayappan, P.: Fast NIC-based barrier over Myrinet/GM. In: Proceedings of the International Parallel and Distributed Processing Symposium. (April 2001)
2. Riesen, R.E., Pedretti, K.T., Brightwell, R., Barrett, B.W., Underwood, K.D., Hudson, T.B., Maccabe, A.B.: The Portals 4.0 message passing interface. Technical Report SAND2008-2639, Sandia National Laboratories (April 2008)
3. Underwood, K.D., Coffman, J., Larsen, R., Hemmert, K.S., Barrett, B.W., Brightwell, R., Levenhagen, M.: Enabling flexible collective communication offload with triggered operations. In: submitted to Proceedings of the 2010 IEEE International Conference on Cluster Computing. (September 2010)
4. Scott, S.L., Thorson, G.: Optimized routing in the Cray T3D. In: PCRCW '94: Proceedings of the First International Workshop on Parallel Computer Routing and Communication, London, UK, Springer-Verlag (1994) 281–294
5. Yih Huang, P.K.M.: Efficient collective operations with ATM network interface support. In: Proceedings of the International Conference on Parallel Processing. (August 1996) 34–43
6. Yu, W., Buntinas, D., Graham, R.L., Panda, D.K.: Efficient and scalable barrier over Quadrics and Myrinet with a new NIC-based collective message passing protocol. In: Proceedings of the Workshop on Communication Architecture for Clusters. (April 2004)
7. Buntinas, D., Panda, D.K., Duato, J., Sadayappan, P.: Broadcast/multicast over Myrinet using NIC-assisted multidestination messages. In: Proceedings of the Fourth International Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing. (January 2000)
8. Wagner, A., Jin, H.W., Panda, D.K., Riesen, R.: NIC-based offload of dynamic user-defined modules for Myrinet clusters. In: Proceedings of the 2004 IEEE International Conference on Cluster Computing. (September 2004) 205–214
9. Underwood, K.D., Levenhagen, M., Rodrigues, A.: Simulating Red Storm: Challenges and successes in building a system simulation. In: 21st International Parallel and Distributed Processing Symposium (IPDPS'07). (March 2007)
10. Hoefler, T., Mehlan, T., Mietke, F., Rehm, W.: Fast barrier synchronization for InfiniBand. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. (April 2006)
11. Ferreira, K.B., Bridges, P., Brightwell, R.: Characterizing application sensitivity to OS interference using kernel-level noise injection. In: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, Piscataway, NJ, USA, IEEE Press (2008) 1–12